DIGITAL SYSTEM DESIGN LABORATORY

LAB 7

MULTI-CYCLE MICROPROCESSOR DESIGN

I. LAB OBJECTIVES

This Lab experiments are intended to design and test a Multi-Cycle Microprocessor

II. DESCRIPTION

Multi Cycle Microprocessor datapath to be implemented is in figure 2.1.

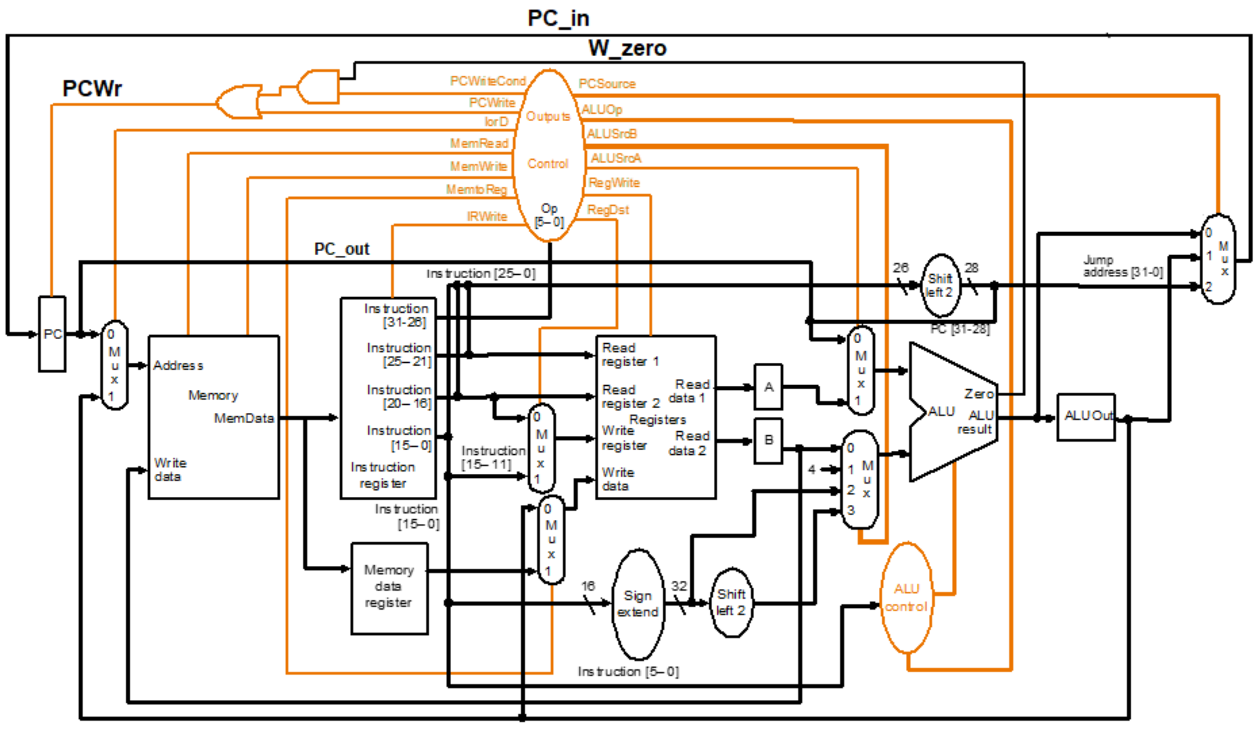


Figure 2.1: Multi-cyclye Cycle Microprocessor DataPath

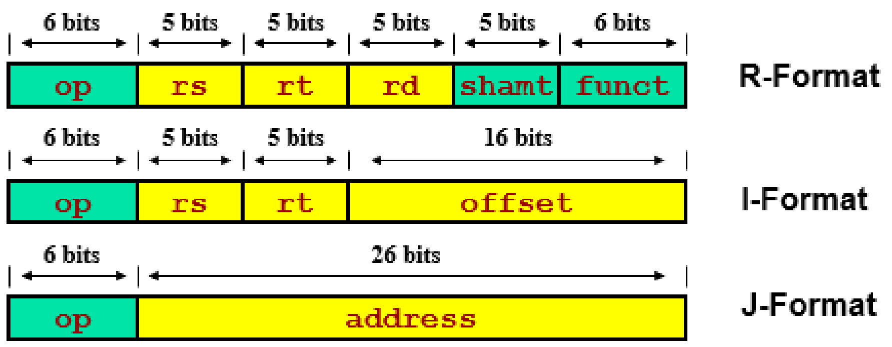
III. LAB PROCEDURE

III.1 EXPERIMENT NO. 1

III.1.1 AIM: To understand and write the assembly codes using MIPS Instruction Instruction Operation codes:

Op	Opcode name	Value
000000	R-format	R-Type
000010	jmp	J-Tpye
000100	beq	I-Type
100011	lw	
101011	sw	
010000	addi	

Instruction Formats:



Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

Register names and orders:

Name	Register number	Usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved (by callee)
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Assume the Assembly code code start from address PC=0x00000000, one instruction is store in one memory location.

Testing Assembly Program 1:

```
          Instruction              Meaning
Begin:    addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2
          addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3
          addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5
          add $s4, $s2, $s3     // $s4 = $s2 + $s3  => R20=0x77
          sub $s1, $s2, $s3     // $s1 = $s2 – $s3  => R17=0x22
          sw $s1, 0x02($s2)     // Memory[$s2+0x02] = $s1
          lw $s6, 0x02($s2)     // $s6 = Memory[$s2+0x02]
          bne $s5, $s4, End     // Next instr. is at End if $s5 != $s4
          addi $s8, $zero, 0x10 // load immediate value 10 to register $s8
          beq $s5,$s4, End      // Next instr. is at End if $s7 == $s4
          addi $s8, $zero, 0x20 // load immediate value 20 to register $s8
End:      j End                 // jump End
```

Testing Assembly Program 2:

```
          Instruction              Meaning
Begin:    addi $s2, $zero, 0x55 // load immediate value 0x55 to register $s2
          addi $s3, $zero, 0x22 // load immediate value 0x22 to register $s3
          addi $s5, $zero, 0x77 // load immediate value 0x77 to register $s5
          add $s4, $s2, $s3     // $s4 = $s2 + $s3  => R20=0x77
          sub $s1, $s2, $s3     // $s1 = $s2 – $s3  => R17=0x22
          sw $s1, 0x02($s2)     // Memory[$s2+0x02] = $s1
          lw $s6, 0x02($s2)     // $s6 = Memory[$s2+0x02]
          beq $s5,$s4, End      // Next instr. is at End if $s7 == $s4
          addi $s8, $zero, 0x10 // load immediate value 10 to register $s8
          bne $s5, $s4, End     // Next instr. is at End if $s5 != $s4
          addi $s8, $zero, 0x20 // load immediate value 20 to register $s8
End:      j End                 // jump End
```

III.1.2 LAB ASSIGNMENT

1) Compile the Assembly **Testing Assembly Program 1** into machine code (decimal code and binary code)

2) What is the value of Register $s8 after running **Testing Assembly Program 1** program

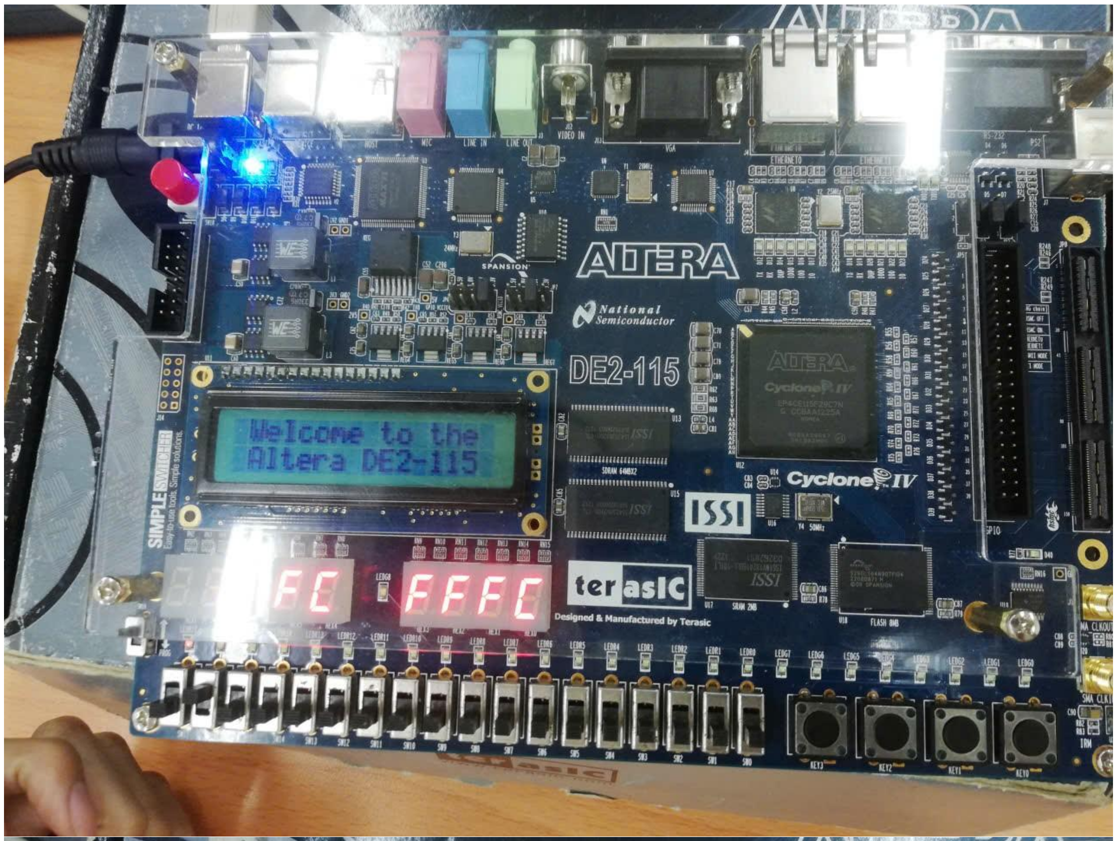
3) Compile the Assembly **Testing Assembly Program 2** into machine code (decimal code and binary code)

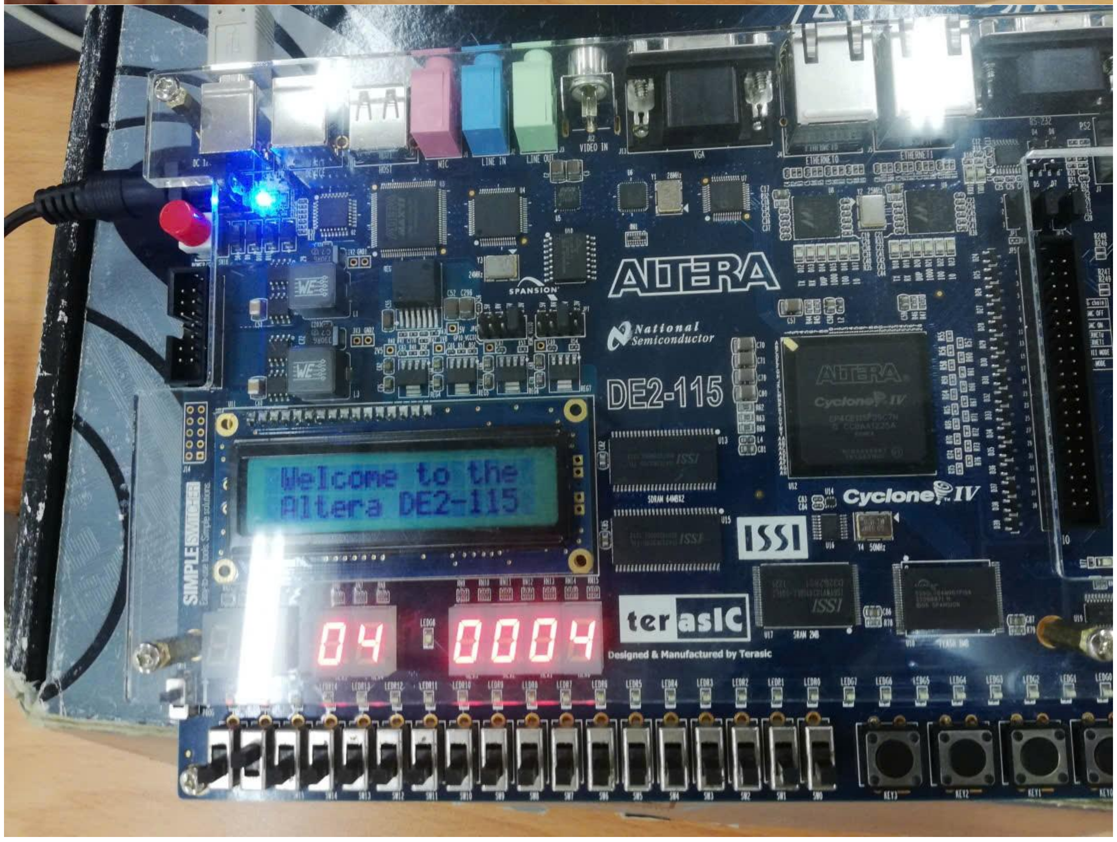
4) What is the value of Register $s8 after running **Testing Assembly Program 2** program

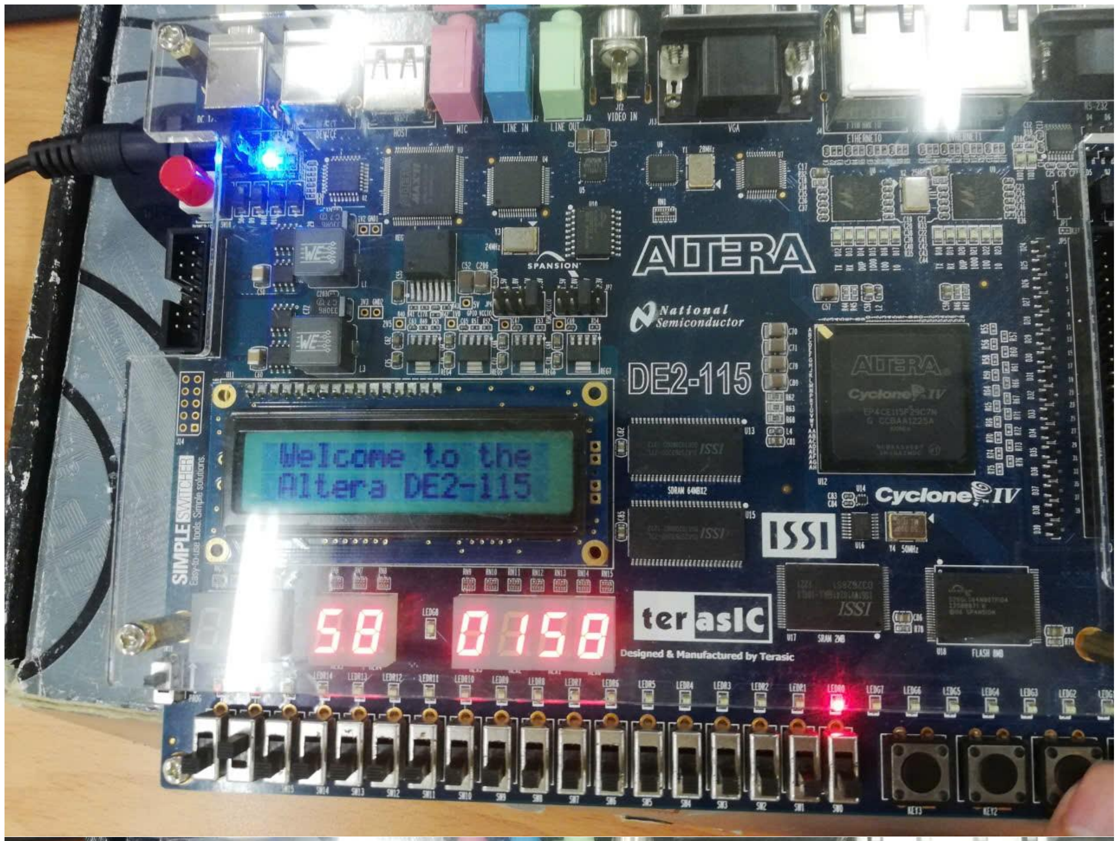
```
module lab7_ex1 (
   input        CLOCK_50,
   input  [3:0] KEY,
   input  [17:0] SW,              // <<< BỔ SUNG SW
   output [16:0] LEDR,
   output [6:0]  LEDG,
   output [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6
);

   // ===================================================
   // Reset (KEY0 – active low)
   // ===================================================
   wire reset = ~KEY[0];

   // ===================================================
   // STEP CLOCK từ KEY1
   // ===================================================
   reg [19:0] db_cnt;
   reg key1_sync0, key1_sync1;
   reg key1_stable, key1_stable_d;

   always @(posedge CLOCK_50) begin
      key1_sync0 <= ~KEY[1];
      key1_sync1 <= key1_sync0;

      if (key1_sync1 == key1_stable)
         db_cnt <= 0;
      else begin
         db_cnt <= db_cnt + 1;
         if (db_cnt == 20'd1_000_000) begin
            key1_stable <= key1_sync1;
            db_cnt <= 0;
         end
      end

      key1_stable_d <= key1_stable;
   end

   wire clk_step = key1_stable & ~key1_stable_d;
```

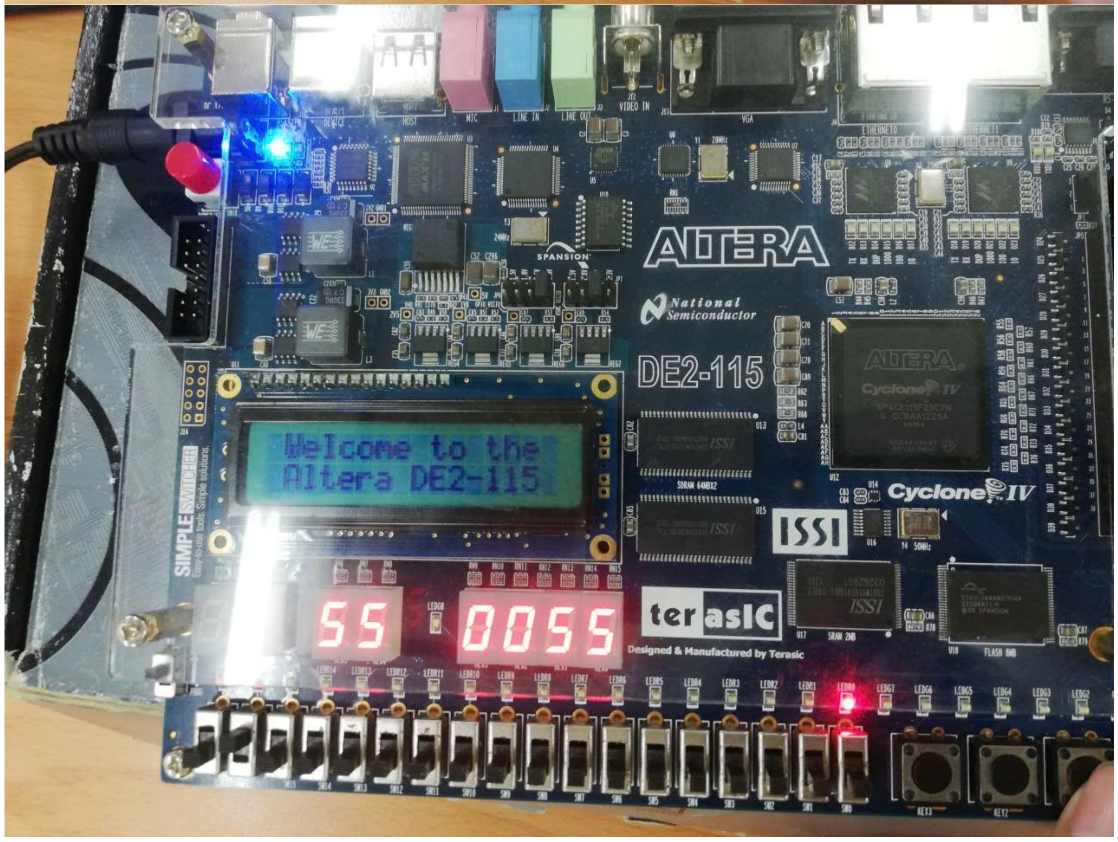
```
// ====================================================
// CPU
// ====================================================
wire [31:0] PC;
wire [31:0] ALUResult;

Datapath_Multi_cycle_Processor U(
    .clk(clk_step),
    .in_reset(reset),

    // các output bạn quan tâm
    .PC_out(PC),
    .ALU_out(ALUResult),

    // các ngõ khác không dùng, có thể bỏ qua hoặc nối hở (*)
    .PCWr(),
    .IRwrite(),
    .MemRead(),
    .PC_in(),
    .Mem_Read_data(),
    .instruction(),
    .MDR_out(),
    .mux_2_out(),
    .B_data(),
    .ALU_in_B(),
    .ALU_in_A(),
    .ALU_out_hold(),
    .Jump_addr(),
    .jump_28_bit()
);

// ====================================================
// LEDR – Program Counter
// ====================================================
assign LEDR[7:0]   = PC[9:2];     // PC / 4 = instruction index
assign LEDR[16:8]  = PC[16:8];

// ====================================================
// LEDG – trạng thái
// ====================================================
assign LEDG[0] = reset;
```

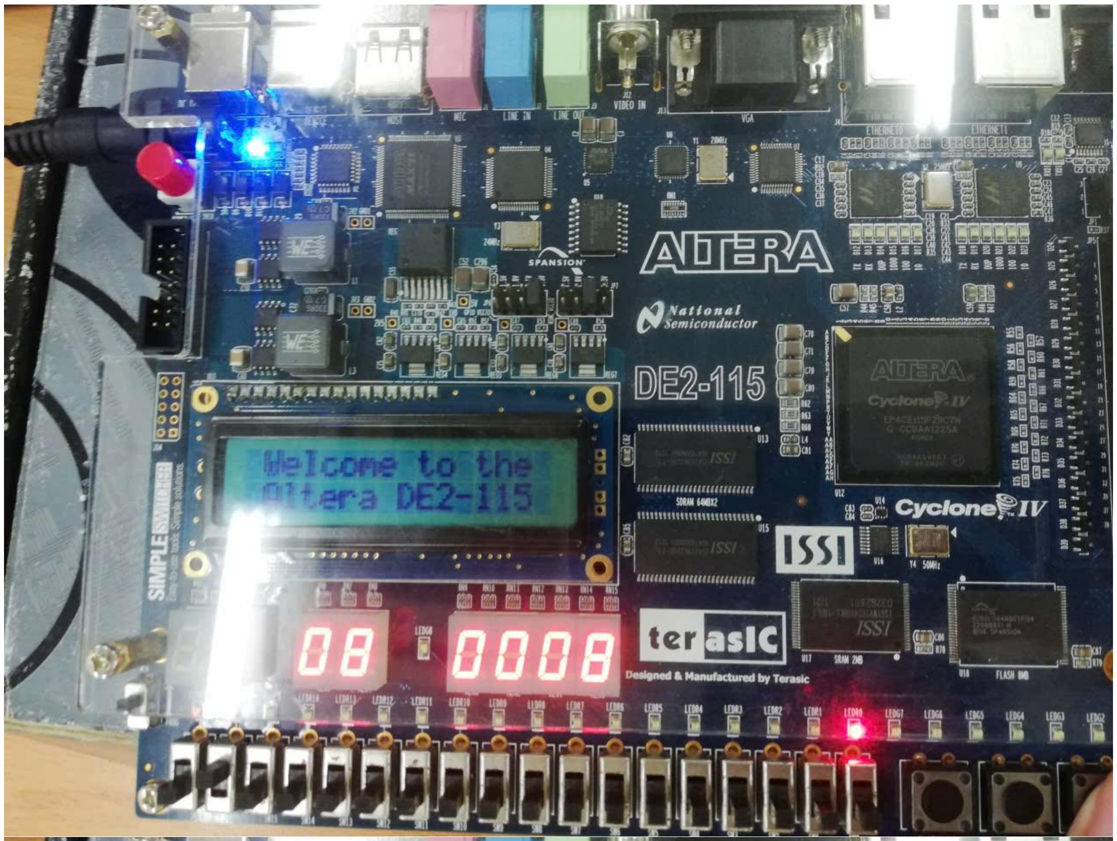
```
    assign LEDG[1] = clk_step;
    assign LEDG[6:2] = 0;

    // ==========================================================
    // MUX HIỂN THỊ BẰNG SW[17:16]
    // ==========================================================
    wire [31:0] disp_bus;

    assign disp_bus =
        (SW[17:16] == 2'b00) ? PC :
        (SW[17:16] == 2'b01) ? ALUResult :
        (SW[17:16] == 2'b10) ? {24'b0, PC[7:0]} :
        PC;

    // ==========================================================
    // HEX – hiển thị disp_bus
    // ==========================================================
    HEX7SEG H0 (disp_bus[3:0],   HEX0);
    HEX7SEG H1 (disp_bus[7:4],   HEX1);
    HEX7SEG H2 (disp_bus[11:8],  HEX2);
    HEX7SEG H3 (disp_bus[15:12], HEX3);

    HEX7SEG H4 (ALUResult[3:0], HEX4);   // giữ nguyên
    HEX7SEG H5 (ALUResult[7:4], HEX5);

    HEX7SEG H6 ({2'b00, SW[17:16]}, HEX6); // hiển thị mode

Endmodule
```

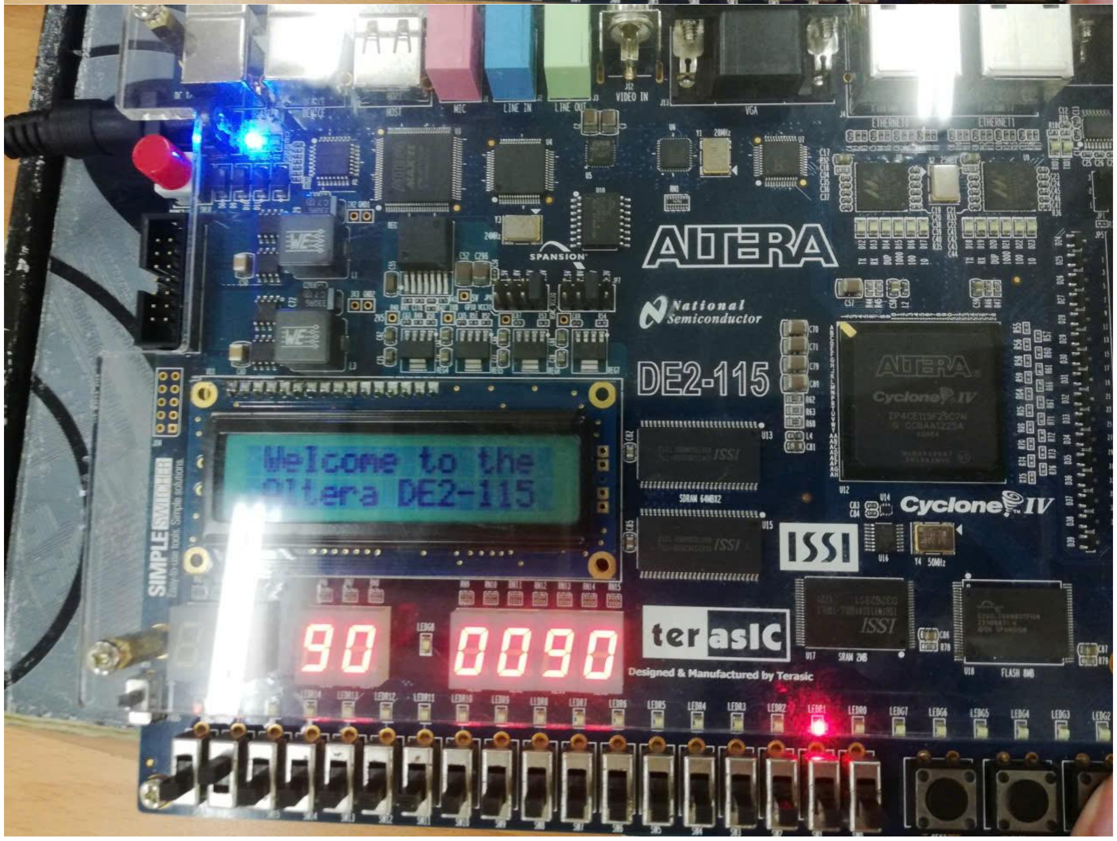

Welcome to the
Altera DE2-115
FC
FFFC
ALTERA
DE2-115
Cyclone IV
terasIC

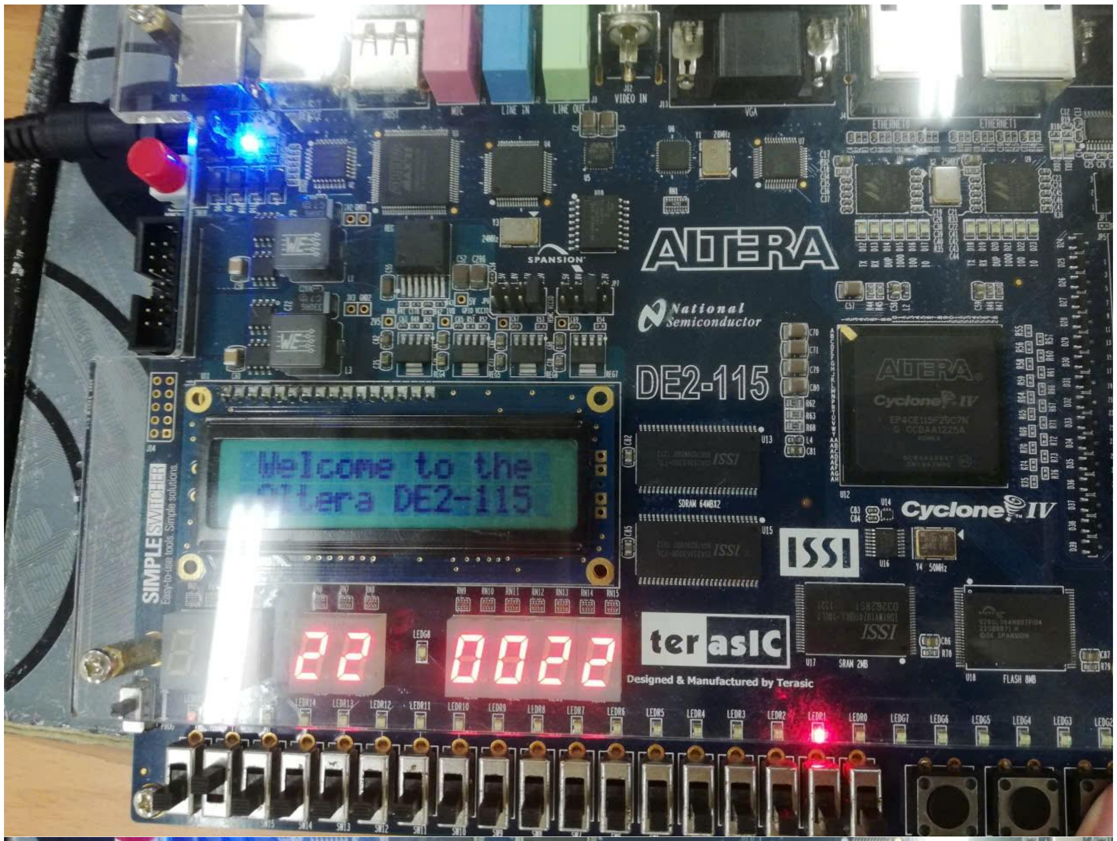

Welcome to the
Altera DE2-115
04
0004
ALTERA
DE2-115
Cyclone IV
terasIC

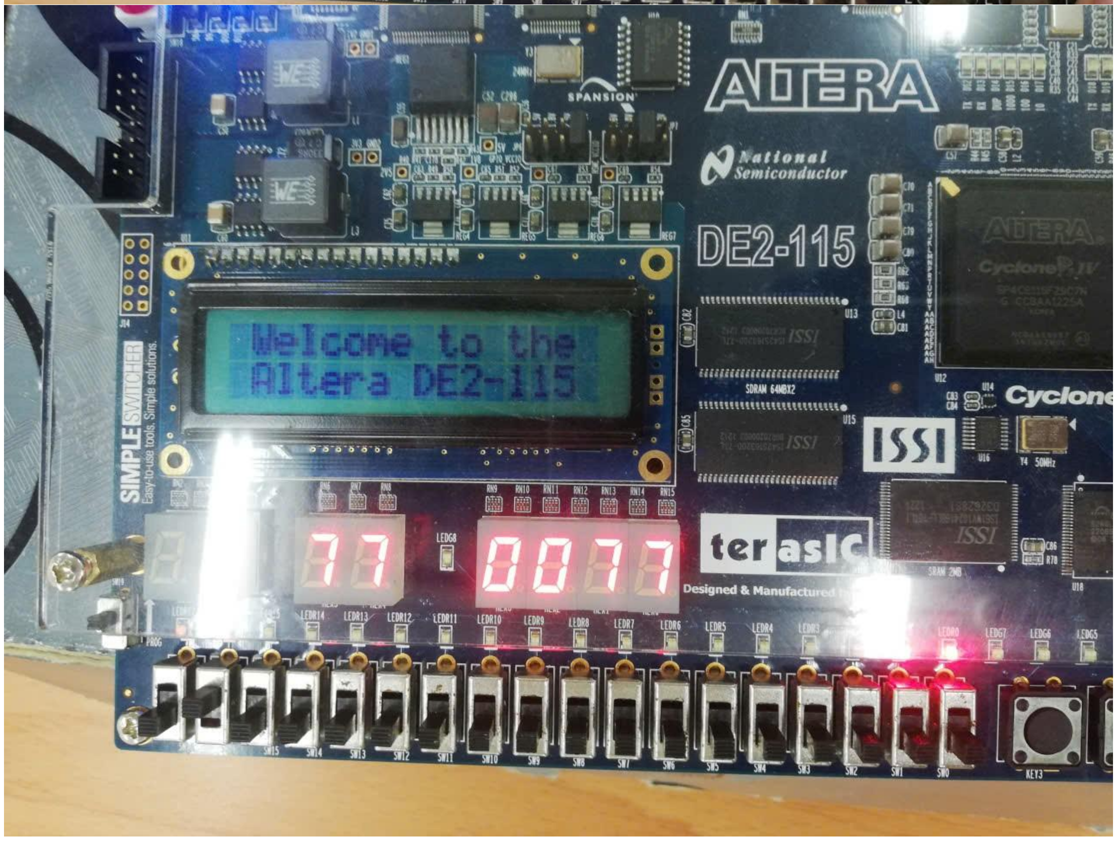

Welcome to the
Altera DE2-115
58 0158


Welcome to the
Altera DE2-115
55 0055


Welcome to the
Altera DE2-115
08
0008
ALTERA
DE2-115
Cyclone IV
terasIC


Welcome to the
Altera DE2-115
90
0090
ALTERA
DE2-115
Cyclone IV
terasIC


Welcome to the
Altera DE2-115
22
0022


Welcome to the
Altera DE2-115
77
0077